VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 Vojtěch Kožuch

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vojtěch Kožuch**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ICT Capital s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Mgr., Ing. Miroslav Kaděra

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 17. dubna 2020 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 17. dubna 2020

.....

Mgr. Ing. Miroslav Kaděra
jednatel, ICT Capital s.r.o.

Mé poděkování patří panu doc. Mgr. Jiřímu Dvorskému, Ph.D. za odborné vedení této práce a společnosti ICT Capital s.r.o. v čele s Mgr. Ing. Miroslavem Kaděrou za poskytnuté zázemí pro zisk mnoha cenných zkušeností.

Abstrakt

Tato práce vznikla jako výstup z mé činnosti ve společnosti ICT Capital s.r.o. v rámci absolvování individuální odborné praxe. Práce zpracovává informace týkající se společnosti, ve které došlo k absolvování praxe, a projektu London Theatre Direct, v rámci něhož byla tvořena veškerá zadání. Obsah je věnován především rozboru zadaných úkolů a jejich vypracování. Závěrem je zhodnocení mého působení ve společnosti. Závěr této práce se také věnuje výčtu zužitkovaných znalostí, získaných za dobu dosavadního studia, a těch, které jsem pro výkon práce postrádal či byly nedostačující.

Klíčová slova: .NET, API, webová aplikace, dodavatelský systém

Abstract

This thesis was created as the output of completion of individual professional practice in ICT Capital s.r.o. It deals with information about the company as well as project London Theatre Direct within which all my assignments were made. The content of this thesis is focused mainly on particular tasks and its processing. The conclusion of this thesis evaluates the practice, mentioning both used knowledge gained from previous studies as well as those I was missing.

Keywords: .NET, API, web application, supply system

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CD	–	Continuous Delivery
CI	–	Continuous Integration
CLR	–	Common Language Runtime
CMS	–	Content Management System
CRM	–	Customer Relationship Management
CSS	–	Cascading Style Sheets
EDM	–	Entity Data Model
ERP	–	Enterprise Resource Planning
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
IoT	–	Internet of Things
LINQ	–	Language Integrated Query
MVVM	–	Model-View-ViewModel
REST	–	Representational State Transfer
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
SŘBD	–	Systém Řízení Báze Dat
URL	–	Unified Resource Locator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato práce je výstupem mé individuální odborné praxe, jež jsem absolvoval ve společnosti ICT Capital s.r.o. Kromě samotného oboru činnosti, jimž se tato společnost zabývá a jenž reflektuje mé zájmy a směr, kterým bych se chtěl dále v rámci svého studia a profese ubírat, mě zaujalo především portfolio této společnosti. Technologická řešení a projekty, kterými se společnost prezentuje, mě přiměly ucházet se o možnost stát se toho všeho součástí. Této společnosti se věnuje ještě následující kapitola.

V ICT Capital jsem v rámci své odborné praxe pracoval na pozici softwarového vývojáře se specializací na vývoj webových aplikací technologií Microsoft .NET. Veškerá má pracovní náplň se odvíjela od prací na projektu jednoho z největších zákazníků, a sice zahraniční společnost London Theatre Direct Ltd. Nejen že jsem dostal příležitost podílet se významnou měrou na vývoji, ale měl jsem zároveň možnost nahlédnout do vedení tak rozsáhlého projektu, komunikace se zákazníkem či procesu testování.

První kapitola této práce se věnuje společnosti ICT Capital. Zaměřuji se na popis technologických řešení, stylu vedení projektů a způsobu organizace práce na nich. Následuje kapitola věnující se projektu London Theatre Direct. Její součástí je mimo jiné výčet několika technologií, s nimiž jsem měl možnost se v rámci tohoto projektu setkat. Kapitola věnována modulům pro připojení dodavatelských systémů definuje některé pojmy, a uvádí tak čtenáře do této problematiky. Následně se v samostatné kapitole zaměřuji na vybraná zadání, která se vztahují k nově implementovanému modulu pro připojení dodavatelského systému SeatGeek, který mě provázel téměř po celou dobu strávenou ve společnosti ICT Capital a na projektu London Theatre Direct. Zadání jsou řazena chronologicky v pořadí, v jakém jsem je vypracovával. V závěru práce zhodnocuji výstupy této odborné praxe. Součástí zhodnocení je výčet zužitkovaných teoretických a praktických znalostí a dovedností, nabytých za dobu mého dosavadního studia, a těch, které mi pro plnění mých pracovních povinností scházely.

2 Společnost ICT Capital

ICT Capital [1] je technologickou firmou sídlící od roku 2011 ve Frýdku-Místku. Společnost se zabývá především zakázkovým vývojem software pro web, desktop i mobilní zařízení. Od roku 2019 využívá na všech svých projektech a pro marketingové účely značku ITIXO. Ve stejném roce firma rozšířila své působiště také o ostravskou pobočku, jež se nachází v budově *Smart Innovation Center Skelet*.

2.1 Technologická řešení

Z pohledu technologických řešení se společnost zaměřuje především na vývoj webových a desktopových aplikací. K tomu využívá primárně technologií platformy společnosti Microsoft. Její portfolio tvoří řada již hotových řešení, mezi něž patří především ERP a CRM systémy či komplexní rezervační systémy a e-shopy. Pro tato řešení společnost často využívá výhod hybridní architektury, která spočívá v kombinaci aplikací běžících na dedikovaných serverech s cloudovými službami.

Mezi zmíněná řešení se řadí aplikace pro sběr a zobrazování velkého množství dat. Takovými řešeními jsou informační systémy pro skladové hospodářství či plánování výroby. Řadí se sem ale i projekty se specifickými požadavky. Společnost tak aktuálně vyvíjí aplikaci zahrnující integraci čteček čárových kódů spolu s dotykovým ovládáním přizpůsobeným pro práci v ochranných pomůckách.

Některé projekty bývají převzaty jako částečné či zastaralé řešení. To s sebou přináší potřebu vypořádávat se s existujícím starším kódem a jeho revitalizací. Firma se obecně snaží svým zákazníkům nabídnout komplexní řešení, které v případě těchto projektů zahrnuje po předchozí analýze možné návrhy modernizace. Mezi tyto projekty se řadí také projekt London Theatre Direct (viz kapitola 3 na straně 18).

Integrace metod pro příjem bezhotovostních plateb patří mezi další z domén této společnosti. Pracovníci mají zkušenosti s integrací platebních metod Apple Pay, Google Pay, Amazon Pay či PayPal. Patří sem ale i metody podporující platbu pomocí kryptoměn.

Sada technologií, které společnost využívá ve svých řešeních, zahrnuje .NET Framework, .NET Core, Vue.js, Node.js, framework DotVVM či služby platformy Microsoft Azure. O některých z těchto technologií se podrobněji zmiňuji v kontextu projektu London Theatre Direct (viz podkapitola 3.3 na straně 22).

2.2 Organizace práce a začlenění

Do společnosti jsem byl přijat na pozici softwarového vývojáře se specializací na vývoj webových aplikací technologií Microsoft .NET. Veškerá zadání vztahující se k projektu London Theatre Direct jsem měl možnost diskutovat nejen s mým konzultantem, který se mi po celou dobu věnoval, ale také v rámci několikačlenného týmu zkušených programátorů, kteří mi poskytli potřebný vhled do samotného projektu.

Pro správu pracovních postupů společnost využívá sadu nástrojů Microsoft Azure DevOps [2]. Kód je pak spravován v repozitářích distribuovaného systému pro správu verzí Git [3], který výrazně usnadňuje a zpřehledňuje kooperaci v rámci vícečlenného týmu. Samotný průběh vývoje je řízen za pomoci vybraných technik iterativního a inkrementálního způsobu řízení vývoje software. Společnost kombinuje techniky agilních metod **Kanban** a **Scrum**. Zmíněným nástrojům a metodikám se blíže věnuje text níže, který zároveň popisuje jejich integraci v kontextu společnosti ICT Capital.

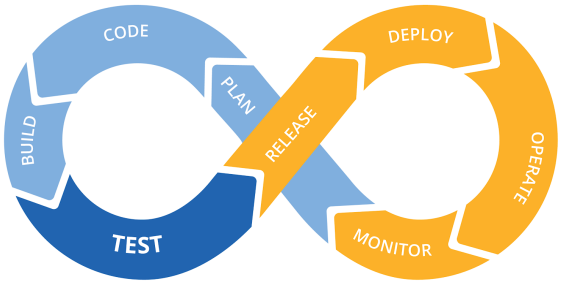
2.2.1 Microsoft Azure DevOps

Sada nástrojů Microsoft Azure DevOps [2] slouží pro správu pracovních postupů. Tato sada moderních služeb zefektivňuje práci, zpřehledňuje postup v rámci projektu a poskytuje nástroje pro testování a nasazování implementovaných řešení. Člení se do několika kategorií. Za nejvýznamnější považuji Azure Boards, Azure Pipelines a Azure Repos.

Azure Boards poskytují prostředí pro plánování, sledování a diskutování práce mezi členy týmů. Společnost ICT Capital pro plánování práce na svých projektech využívá **karty Kanban**, které slouží k plánování takzvaných *sprintů* o délce trvání zpravidla dvou týdnů. Ty slouží především k definování jasného časového rámce pro splnění zadaných úkolů. *Backlog* si pak lze představit jako úložiště pro uskladnění uzavřených a plánovaných úkolů. Karty Kanban jsou realizovány za pomocí tabulky, v niž řádky jsou tvořeny jednotlivými úkoly a sloupce představují fázi, ve které se daný úkol právě nachází. Ve společnosti ICT Capital těmito fázemi jsou: nově vytvořený (*new*), aktivní (*active*), čekající na schválení (*pending approval*) a uzavřený (*closed*). Na konci každého z těchto sprintů se tým programátorů sejde ke společnému vyhodnocení. To zajišťuje obecný přehled všech zúčastněných o pracovní náplni ostatních členů týmu a pokroku v rámci projektu. Výstupem tohoto setkání je zároveň naplánování úkolů pro příští iteraci.

Azure Pipelines umožňují průběžně sestavovat, testovat a nasazovat jednotlivé aplikace. Kromě nástrojů pokrývajících základní scénáře jsou vývojářům poskytnuty pokročilejší služby podporující například kontejnerizaci.

Se sadou nástrojů Azure Pipelines se pojí především metodiky pro zefektivnění samotného sestavení, testování a nasazení software **CI/CD**. Při aplikování *Continuous Integration* (CI) je kód vývojáři udržován ve sdíleném repozitáři, odkud dochází k jeho pravidelnému sestavování a testování za pomocí automatizovaných testů. Tím je zajištěno, že změny v daném repozitáři neporušují stávající funkcionalitu a je možné jejich nasazení. *Continuous Delivery/Continuous*



Obrázek 1: Principy CI/CD [4]

Deployment (CD) často koreluje s CI. Při aplikaci této sady praktik dochází k automatizovanému nasazení změn v kódu do testovacího a následně produkčního prostředí. Životní cyklus software při integraci CI/CD metodik zachycuje obrázek 1.

S těmito metodikami a jejich integrací za pomocí sad nástrojů Azure Pipelines je možné se setkat dnes již téměř na všech aktivních projektech společnosti ICT Capital.

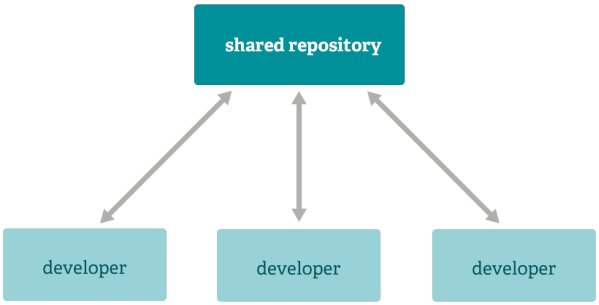
Nástroje **Azure Repos** se vztahují k samotnému kódu, uloženému v repozitářích verzovacího systému, a jeho správě. Na projektech společnosti ICT Capital tímto systémem je Git (viz podkapitola 2.2.2).

Azure Repos umožňují definovat zásady vedoucí ke zvyšování kvality kódu. V projektu tak lze například vynucovat revizi kódu jiným vývojářem či absolvování testů před dokončením žádosti o sloučení změn (anglicky *pull request*).

2.2.2 Git

Git [3] je distribuovaným systémem pro správu verzí navrženým pro využití na projektech různého rozsahu. Důraz je kladen především na rychlost a efektivitu. Jedná se o *open-source* řešení. Základním stavebním prvkem jsou větve, v rámci nichž je kód spravován. Podstata je ve vzájemné nezávislosti těchto větví, sdílení jejich obsahu a jeho snadného sloučení. Git přináší hned několik možností, jak lze tyto větve kódu využít:

- Větve kódu zastávající speciální roli. Lze takto definovat větve, jejichž kód je vyhrazen k produkčnímu nasazení či pro testování.
- Větve kódu vztahující se k dané funkcionalitě. Právě vyvíjenou funkcionalitu lze oddělit do samostatné větve, dokud nedojde k jejímu sloučení, k čemuž zpravidla dochází při dokončení dané funkcionality.
- Větve obsahující experimentální kód. Vzhledem k nezávislosti na jiných větvích lze v každé z nich provádět libovolné úpravy a experimentovat, aniž by kdokoliv jiný cokoliv zpozoroval.



Obrázek 2: Centralizovaná varianta distribuovaného systému Git [3]

Jednou ze základních vlastností tohoto systému je jeho distribuovanost. Dochází tak k opakovanému klonování celého distribuovaného repozitáře, kde každá tato kopie je v podstatě zároveň jeho zálohou, což přispívá ke zvýšení spolehlivosti a usnadnění případného obnovení.

Z výše uvedeného vyplývá, že Git přináší uživatelům výraznou míru flexibility. Lze tak definovat různé pracovní postupy. Tím nejpopulárnějším je centralizovaný (viz obrázek 2). Uživatelé pracují nad sdíleným repozitářem. Pokud chce uživatel provést zápis do sdíleného repozitáře, musí mít propsány všechny změny z něj ve své lokální kopii. Tento postup je využíván také ve společnosti ICT Capital.

2.3 Společnost Riganti a framework DotVVM

ICT Capital spolupracuje s řadou významných partnerů. Patří mezi ně také technologická společnost Riganti s.r.o. [5]. Tato společnost sídlící v Praze a Brně stála při zrodu platformy DotVVM [6].

DotVVM je framework pro tvorbu webových aplikací. Pro usnadnění vývoje poskytuje celou řadu nástrojů a předpřipravených komponent. Využívá návrhového vzoru MVVM a minimalizuje nutnost znalosti jazyka JavaScript. Ve valné většině případů vývojáři postačí znalost programovacího jazyka C#, značkovacího jazyka HTML a kaskádových stylů CSS. Tento framework lze považovat za alternativu k již nevyvíjenému ASP.NET Web Forms (viz podkapitola 3.3.2 na straně 22) společnosti Microsoft.

2.4 Společnost Update Conference

Společnost Update Conference s.r.o. je dalším z řady významných partnerů ICT Capital. Aktuálně nejvyznamnějším počinem této společnosti je pravidelně konaná konference *Update Conference Prague* [7]. Návštěvníkům je každoročně umožněno v rámci dvou konferenčních dnů navštívit více než 40 přednášek odborníků z Evropy, Spojených států amerických a dalších zemí. Pracovníci firmy ICT Capital se aktivně podílejí na přípravě této konference.

3 Projekt London Theatre Direct

Společnost London Theatre Direct [8] se zabývá především distribucí a prodejem vstupenek vztahujících se k různým kulturním událostem. Primárně se jedná, jak samotný název společnosti napovídá, o prodej vstupenek na představení londýnských divadel. Patří sem ale i vstupenky na různé koncerty či turistické atrakce pořádané v Londýně. Společnost se zároveň snaží nabízeným sortimentem expandovat i do jiných zemí, a tak v rámci jedné ze svých platforem již začala nabízet vstupenky také na představení divadel amerických. London Theatre Direct vnímám jako agilní společnost, která podporuje moderní technologie, je flexibilní vůči změnám trhu a pro úspěch se nebojí experimentovat, díky čemuž projekt samotný skýtá příležitosti k seznámení se s řadou zajímavých technologií.

Zaujaly mě především interaktivní plány hlediště a zobrazování obsazenosti sedadel v reálném čase. Jejich podobu zachycuje obrázek 11 (strana 45, příloha A). Další doménou projektu je podpora široké škály platebních metod. Patří sem Apple Pay, Google Pay či PayPal. Společnost v minulosti nabízela dokonce možnost platby pomocí kryptoměn. Kromě e-shopu a jiných aplikací s nim spjatých společnost nabízí celou škálu nástrojů pro integraci partnerských systémů. Těm se věnují následující odstavce.

3.1 Řešení pro integraci partnerských systémů

Jednou z významných domén této platformy jsou řešení pro integraci partnerských systémů. Společnost poskytuje za jistou provizi již hotové řešení pro distribuci a prodej vstupenek. Dává možnost svým partnerům integrovat v krátkém časovém horizontu komplexní systém s celou řadou možností personalizace. Patří sem úpravy vztahující se ke vzhledu webové stránky či nastavení týkající se cenotvorby.

S rostoucím počtem partnerům tato platforma expanduje, čímž se zároveň zvyšuje její vliv a podíl na trhu. Níže popsaná řešení, která jsou partnerům nabízená, lze zároveň kombinovat.

3.1.1 Whitelabel řešení

Whitelabel řešení je nejpopulárnější. Partnerům je poskytnut systém, který na pozadí využívá totožných technologií jako samotný e-shop společnosti. Mezi možnosti personalizace patří volba barev, vlastního loga či nastavení týkající se analytických nástrojů společnosti Google.

3.1.2 API řešení

Přístup skrze rozhraní API je dalším ze způsobů integrace partnerského systému. Preferovanou možností je napojení skrze REST API. Poskytováno je však také rozhraní SOAP API. Pro co nejjednodušší integraci poskytuje společnost vývojářům veřejný prohlížeč k otestování volání jednotlivých API požadavků a zároveň je pro ně k dispozici dokumentace [9]. Veřejné webové rozhraní API umožňuje zreplikovat kompletní proces nákupu vstupenek, od získání dostupnosti

pro dané představení přes vytvoření košíku a vložení inventáře do něj až po dokončení objednávky a přesměrování na platební bránu pro zaplacení.

3.1.3 B2B řešení

Toto řešení je vhodné pro partnery realizující obchodní vztahy, při nichž dochází k přímému styku s koncovým zákazníkem. Příkladem mohou být cestovní agentury nebo recepce hotelů.

Jedná se o samostatnou webovou aplikaci ASP.NET Web Forms, která uživatelům nabízí přívětivé rozhraní pro vytváření objednávek v reálném čase. Patnerům je mimo jiné nabídnuta možnost správy vícero uživatelských účtů, což jim umožňuje spravovat provize z prodejů provedených jednotlivými zaměstnanci.

3.1.4 WordPress plugin

Posledním ze zmíněných řešení pro integraci partnerských systémů je rozšíření pro WordPress. Jedná se o populární sotfware, řadící se mezi takzvané CMS systémy, pro tvorbu webového obsahu.

Rozšíření společnosti London Theatre Direct umožňuje do takto vytvořené webové stránky zahrnout jí nabízené produkty, z jejichž prodeje majitel získává provizi. Samotná integrace je snadná, jelikož se většina operací odehrává na pozadí. Partnerům jsou i zde nabídnuty možnosti personalizace, které jim umožňují lépe zakomponovat nabídku do kontextu dané webové stránky.

3.2 Struktura aplikací

Projekt do své správy společnost ICT Capital převzala před několika lety jako již rozpracované řešení. V porovnání se stávající podobou projektu se však jednalo pouze o zlomek funkcionality. I přesto podstatná část původního řešení zůstala v projektu zakořeněna, a vývojáři se tak dodnes musí vypořádávat se zastaralým kódem (anglicky *legacy code*), který se postupně daří odstraňovat a projekt samotný modernizovat.

Mezi nejzásadnější počiny poslední doby se řadí zavádění vyvažování zátěže (anglicky *load balancing*) aplikací běžících na vícero hostujících zařízeních, což zároveň přináší jejich bezvýpadkové nasazování do produkčního prostředí. Z pohledu zdrojového kódu je důraz kladen především na jeho kvalitu. Ke zlepšení jednotnosti zdrojového kódu dopomáhají obecné zásady definující způsob psaní. Velice pozitivně na mě působí, že společnost i přes nově se objevující požadavky klienta, na které musí být schopná flexibilně reagovat, dokáže projekt zároveň technologicky modernizovat, aby tak plnil parametry moderního systému. Spolu s podpůrnými aplikacemi se platforma sestává z vyšších desítek aplikací. Následující odstavce nastiňují pouze ty nejpodstatnější z nich.

3.2.1 Webová stránka London Theatre Direct

Webová stránka London Theatre Direct je nejkritičtějším prvkem této platformy. Jedná se o webovou aplikaci vyvíjenou na platformě ASP.NET Web Forms. Jejím jádrem je integrovaný systém umožňující rezervaci a nákup vstupenek na vybraná představení londýnských divadel, koncerty či turistické atrakce. Důraz je kladen na interaktivnost, intuitivnost a jednoduchost. V nejčastějším scénáři uživatel vybere událost, pro kterou chce vstupenky zakoupit, zvolí konkrétní datum konání a vybere sedadla. Následně je přesměrován na stránku pro zadání potřebných údajů k objednávce a provedení platby s pomocí jedné z nabízených platebních metod.

Podobu domovské stránky této aplikace zachycuje obrázek 10 (strana 45, příloha A). Kromě rezervačního systému je zde řada obsahových stránek. Tyto stránky jsou věnovány informacím o jednotlivých divadlech a událostech. Samostatnou sekci tvoří články zpracovávající informace k dění okolo divadelní scény.

3.2.2 Administrace

Tato webová aplikace, vyvíjená opět na platformě ASP.NET Web Forms, je užívaná primárně zaměstnanci kontaktního centra. Je členěná do několika sekcí.

V sekci *Book tickets* lze provádět objednávky podobně jako v případě jiných aplikací této platformy. Sekce *Site* slouží ke správě webového obsahu nejen pro obsahové stránky. V sekci *Sales* se mimo jiné nachází výkazy o prodejích. Zobrazení přehledu aktivních partnerů s možností editace jejich údajů a nastavení, umožňuje sekce *Partners*. Nástroje a nastavení vztahující se k modulům pro připojení dodavatelských systému (viz kapitola 4 na straně 24) sdružuje sekce *External integrations*.

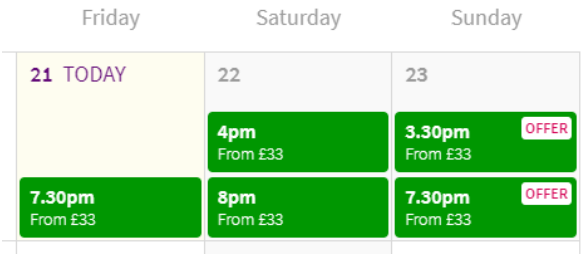
Pro přístup do tohoto systému je vyžadována autentizace podporující dvoufaktorové ověření s definovanými uživatelskými rolemi.

3.2.3 Aplikace pro správu dat v mezipaměti

Spousta dat je v projektu pro zefektivnění načítání ukládána do mezipaměti, kterou je zapotřebí pravidelně aktualizovat. U jednotlivých představení těmito informacemi jsou:

- informace vztahující se k jednotlivým sedadlům a jejich dostupnosti,
- celkový počet dostupných a na sebe navazujících sedadel,
- nejnižší částka pro pořízení vstupenky,
- informace o výskytu slevové nabídky a její případná nejvyšší možná výše.

Informace o jednotlivých sedadlech jsou závislé na dodavatelském systému, ke kterému se dané představení vztahuje (viz kapitola 4 na straně 24). Proto pro každý z těchto systémů existuje aplikace, která tato data s externím systémem synchronizuje.



Obrázek 3: Položky kalendáře obsahující informace k představením

Zbylé informace jsou nezávislé na systému dodavatele a jsou spravovány samostatnou aplikací. Využití těchto informací na stránce s kalendářem v rámci webové stránky London Theatre Direct zachycuje obrázek 3.

K synchronizaci těchto dat dochází na základě požadavků přijímaných aplikacemi na bázi služby Azure Service Bus (viz podkapitola 3.3.5).

3.2.4 Aplikace zpracovávající data k objednávkám

V rámci procesu vyřizování objednávek dochází po přijetí platby k několika úkonům, jenž jsou prováděny asynchronně. Jedná se o dokončení objednávky na straně dodavatelského systému, odeslání potvrzovacího e-mailu či vygenerování souborů se vstupenkami k vytisknutí. Každý z těchto úkonů je obsluhován samostatnou aplikací a požadavky jsou vyřizovány skrze databázi.

3.2.5 B2B a webové rozhraní API

O těchto aplikacích pojednává již kapitola 3.1 ve vztahu k integraci partnerských systémů.

3.3 Použité technologie

3.3.1 Microsoft .NET

Microsoft .NET [10] je platforma pro vývoj různých typů aplikací. Poskytuje jednotné prostředí pro tvorbu webových, mobilních i desktopových aplikací. Mezi pokročilejší oblasti Microsoft .NET patří podpora IoT, strojového učení či nástrojů pro tvorbu počítačových her. Pod samotným pojmem si lze představit sadu nástrojů, knihoven a programovacích jazyků, s jejichž pomocí lze tyto aplikace vyvíjet.

.NET Framework je původní implementací, která podporuje běh aplikací v prostředí Windows. Sestává se ze dvou základních komponent:

- *Common Language Runtime* (CLR) se stará o samotný běh aplikace. Mimo jiné zajišťuje správu vláken, zpracovávání výjimek či typovou kontrolu.
- *Class Library* zajišťuje potřebné struktury a funkce. Poskytuje datové typy například pro ukládání a práci s textovými řetězci, daty nebo čísly. Dále zajišťuje operace čtení a zápisu do souborů, připojení k databázi či nástroje pro kreslení.

.NET Core je multiplatformní implementací, jež podporuje běh aplikací nejen v prostředí operačního systému Windows, ale i Linux či macOS. .NET Core je zároveň distribuován pod licencí *open-source*, a tak má kdokoliv možnost nahlédnout či přispět do zdrojových kódů této platformy.

Xamarin/Mono je další implementací této platformy, která se soustředí na mobilní zařízení a vývoj aplikací na ně. Podporuje všechny majoritní operační systémy v této oblasti, mezi něž se řadí Android či iOS.

3.3.2 ASP.NET Web Forms

ASP.NET Web Forms [11] poskytuje platformu pro tvorbu webového obsahu s přístupem připomínajícím spíše vývoj desktopových aplikací. Interakce uživatele s rozhraním systému je založena na vyvolávaní událostí, které jsou dále zachytávány a zpracovávány serverovou částí aplikace.

Framework poskytuje celou škálu již připravených komponent uživatelského rozhraní. Řada z nich odpovídá základním HTML elementům a zapouzdřuje funkce navíc, které by jinak musel vývojář implementovat sám. Některé z komponent přináší pokročilejší funkcionalitu, jako je zpracovávání tabulkových dat. Vývojáři je zároveň poskytnuta možnost vytváření komponent vlastních. Platforma dále výrazně zjednodušuje přenos dat mezi serverovou a klientskou částí aplikace. Vývojář si tak vystačí s minimálními znalostmi vývoje klientské části aplikace.

Tento framework je již však považován za zastaralý a společností Microsoft nepodporovaný. Tato skutečnost znamená, že v něm lze vyvíjet pouze aplikace pro platformu .NET Framework, nikoliv pro později představený .NET Core, který podporuje multiplatformnost, a aplikace v něm psané tak lze spouštět kromě operačního systému Windows také v prostředí Linux či macOS.

3.3.3 Microsoft SQL Server

Microsoft SQL Server se řadí mezi systémy řízení báze dat (SŘBD), mezi něž jsou obecně řazená softwarová vybavení, jež zpřostředkovávají data z perzistentního úložiště (databáze) aplikacím. Pro práci s těmito daty je využíván jazyk SQL a jeho rozšíření *Transact-SQL*, který do jazyka přináší podporu programovacích konstrukcí.

3.3.4 Entity Framework a LINQ

Entity Framework [12] je framework zajišťující objektově-relační mapování. Umožňuje vývojářům přistupovat k databázovým entitám skrze objekty .NET. Odstiňuje je od operací zajišťujících komunikaci a přenos dat a poskytuje dostatečnou míru abstrakce, čímž práci s databází značně zjednodušuje. Základem je *Entity Data Model* (EDM), který provádí mapování doménových tříd (entit) na databázové schéma.

Language Integrated Query (LINQ) [13] je sada technologií umožňující psaní dotazů v jazyce C# .NET. Jedná se o jazyk pro dotazování, jehož hlavním cílem je sjednocení syntaxe přístupu k datům nezávisle na jejich zdroji. Ve spojení s technologií Entity Framework skrze něj lze psát dotazy, které jsou automaticky překládány a prováděny v jazyce SQL.

3.3.5 Azure Service Bus

Azure Service Bus [14] je multitenantní technologie seskupující služby pro zajištění komunikace mezi aplikacemi.

Queues neboli fronty patří mezi nejvíce využívané typy služeb v projektu London Theatre Direct. Slouží jako vrstva pro zprostředkování zpráv k jejich pozdějšímu přijetí a zpracování. Zpráva zveřejněná vybranou aplikací je uložena do fronty, aby později mohlo dojít k jejímu vyjmutí a zpracování příjemcem.

Topics slouží pro jednosměrnou komunikaci. Zpráva zveřejněná vybranou aplikací je zpracováváná zpravidla několika posluchači, kteří tak například mohou reagovat na danou událost, kterou může zveřejněná zpráva signalizovat.

Využití této technologie především usnadňuje škálování. Pokud například existuje fronta a příjemce, který zprávy z ní přijímá a zpracovává, může být aplikace představující příjemce nasazena v několika instancích, čímž bude docházet k paralelnímu zpracovávání při větším zahlcení fronty.

4 Moduly pro připojení dodavatelských systémů

4.1 Dodavatelský systém

Vzhledem k podstatě projektu společnosti London Theatre Direct, jež je založená na distribuci a prodeji vstupenek na různé kulturní události, je zapotřebí, aby platforma dokázala spolupracovat s dodavateli těchto vstupenek. Těmi jsou divadla, řetězce kin, koncertní haly či produkční společnosti. Jako dodavatelský je pak označován informační systém, který těmto dodavatelům poskytuje prostředky pro cenotvorbu, rezervaci a prodej vstupenek. V případě divadel se jedná především o pokladní a rezervační systémy v nich.

Vzhledem k majoritnímu podílu divadelních představení v nabídce produktů platformy London Theatre Direct budu v následujících odstavcích popisovat dodavatelské systémy právě v kontextu divadel. V případě jiných, podobných institucích je situace téměř totožná.

4.1.1 Motivace

Cílem dodavatelských systémů je zefektivnění spolupráce mezi přeprodejci a divadly. Vztah mezi těmito dvěma subjekty je oboustranně prospěšný. Pro divadelní společnost to znamená zvýšení prodejů a rozšíření základny návštěvníků. Přeprodejci pak benefitují z provizí, které pro ně plynou z jednotlivých prodejů.

Dodavatelský systém divadlu kromě analytických dat poskytuje sadu nástrojů pro správu nabídky představení jednotlivých divadelních her. Zásadní je však sdílení dostupnosti sedadel v reálném čase skrze poskytnuté veřejné webové rozhraní API. To zefektivňuje prodej a umožňuje jeho paralelizaci. Před příchodem technik pro sdílení zmiňovaných dat v reálném čase byl vztah přeprodejců s divadly značně neefektivní, jelikož přeprodejce dostával k dispozici od divadla vždy jen omezené množství alokovaných vstupenek, jež mu byly nabídnuty k prodeji a po jejichž vyprodání musel divadlo žádat o další. Divadelní společnost zároveň riskovala, že přeprodejci nabídne těchto vstupenek příliš, čimž ji vznikne ztráta v podobě ušlého zisku.

4.2 Integrace dodavatelského systému

Projekt London Theatre Direct sjednocuje a integruje hned několik dodavatelských systémů. Každá taková integrace reprezentuje samostatný modul, který má pro každý z dodavatelských systémů za úkol především:

- Synchronizaci dat o představeních a vstupenkách. Patří sem cenové hladiny nabízených vstupenek, rozvržení hlediště, ale především obsazenost sedadatel, která je aktualizována v reálném čase.
- Provedení rezervace zákazníkem. K propsání do dodavatelského systému dochází v reálném čase, systém tak okamžitě eviduje daná sedadla jako rezervována a dále je nenabízí, pokud však nedojde k jejich uvolnění.

- Provedení dokončení objednávky a propsání osobních údajů zákazníků. Veškeré rezervace jsou po přijetí platby asynchronně potvrzovány na straně dodavatelského systému.
- Poskytování sady podpůrných nástrojů. Mezi tyto nástroje patří ty pro nastavení slevových nabídek, vložení nově nabízených představení či správu informací o typech sedadel.

Některé moduly pro připojení dodavatelského systému navíc implementují pokročilý režim zvaný *offline booking*. V praxi to znamená, že v případě výpadku systému dodavatele (ať už plánovaného, či nikoliv) je uživatelům nadále umožněno pořizování vstupenek z daného systému. Modul však v tomto režimu nekomunikuje se systémem dodavatele. Pracuje pouze s naposledy aktualizovanými daty v mezipaměti. Rezervace a potvrzování objednávek v tu chvíli také nejsou propisovány do systému dodavatele. K tomu dochází naráz ve chvíli, kdy dojde ke znovuzprovoznění externího systému.

S tímto režimem se pojí riziko možného vzniku takzvaných zdvojených objednávek, jak jsou popisovány situace, kdy si více zákazníků objedná vstupenku pro stejné sedadlo. K těmto situacím dochází vzhledem k tomu, že v takzvaném *offline* režimu nejsou data o dostupnosti aktualizována. Vzniklé zdvojené objednávky poté pracovníci kontaktního centra řeší se zákazníky individuálně.

5 SeatGeek a řešení úkolů

SeatGeek [15] je platforma primárně se zaměřující na prodej a distribuci vstupenek na živá představení původně pouze ve Spojených státech amerických a Kanadě. Jedná se především o sportovní utkání či hudební vystoupení. Společnost se rozhodla s tímto projektem expandovat také do Evropy, a společnosti London Theatre Direct se tak naskytla možnost integrace dalšího z dodavatelských systémů.

Modul pro připojení tohoto dodavatelského systému, na němž jsem pracoval spolu s dalšími vývojáři v rámci několikačlenného týmu, mě provázel téměř po celou dobu mé odborné praxe. Následující úlohy popisují ucelené části tohoto modulu, které jsem vypracovával. Není-li uvedeno jinak, všechny aplikace byly vyvíjeny na platformě .NET Framework 4.8 v jazyce C# 7.3.

Časová náročnost

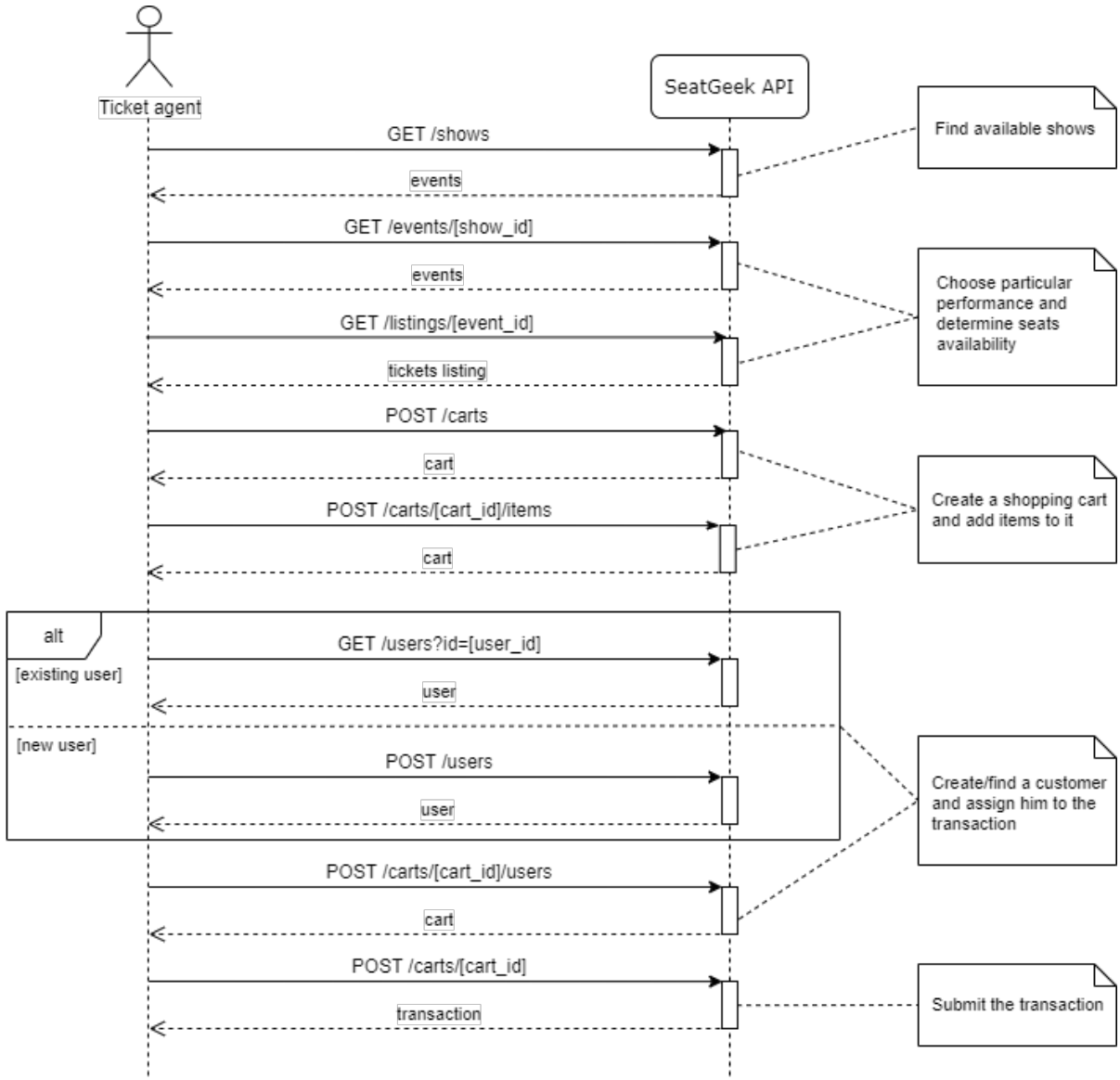
Tabulka 1: Časová náročnost jednotlivých úkolů

Úkol	Počet pracovních dní
Analýza webového rozhraní API	13
Integration Tester	součást analýzy
External Connector	4
Performances Mapper	7
Inventory Synchronizer	15

5.1 Analýza webového rozhraní API

Jádrem komunikace s tímto dodavatelským systémem je veřejné webové rozhraní REST API. Vzhledem k tomu, že znalost tohoto API byla pro samotnou implementaci modulu zcela stěžejní, musel jsem provést jeho analýzu. Společnost SeatGeek pro tyto účely zpracovala podrobnou dokumentaci, která pro mě sloužila jako primární zdroj informací při seznamování se s tímto systémem, a celý proces analýzy a integrace značně usnadnila. Jednalo se především o nalezení základního scénáře vedoucího k vytvoření objednávky. Výsledek této části analýzy zachycuje sekvenční UML diagram na obrázku 4.

Předmětem analýzy nebylo pouze seznámení se s tímto systémem a pochopení základních principů komunikace s ním, ale i snaha o nalezení co nejvíce společných rysů se systémy, pro které již byly moduly v minulosti implementovány. Cílem bylo minimalizovat množství nově vzniklých částí projektu specifických pro tento systém a znovupoužít co největší objem kódu, popisující společné chování s jinými systémy dodavatelů.



Obrázek 4: Sekvenční UML diagram SeatGeek API

5.1.1 Základní pojmy

Domain – instituce zaštiťující konání dané události (například divadlo).

Show – událost, ke které se vztahují jednotlivá představení.

Event – konkrétní představení dané události.

Listing – výpis dostupnosti sedadel konkrétního představení.

Cart – ekvivalent nákupního košíku sloužící pro sdružování inventáře v rámci objednávky.

User – zákazník provádějící danou objednávku.

5.1.2 Autentizace

Uživatel je ověřován na základě hodnot *ID* a *secret*, jež mu jsou poskytnuty před samotným užíváním API, jsou neměnné a specifické pro daného uživatele. Samotné ověření má pak možnost uživatel provést dvěma způsoby:

1. *Query string* parametr. Uživatel hodnoty uvede jako parametry při volání požadavků API v URL.
2. HTTP autentizace. Uživatel hodnoty vloží do hlavičky HTTP požadavku.

5.2 Integration Tester

5.2.1 Specifikace požadavků

Webová aplikace mající za cíl otestovat samotnou komunikaci s webovým rozhraním API systému dodavatele a poskytnout prostředí pro zinscenování základních scénářů práce s nim. Každé volání metody rozhraní API bude v uživatelském rozhraní graficky odděleno do samostatného bloku. Tento blok bude obsahovat formuláře pro zadání vstupních parametrů dané metody. Uživatelské prostředí aplikace dále zobrazí podrobné informace k právě volanému požadavku, včetně informací o případné výjimce, ke které při volání dojde. Samotná data navrácená z API pak budou zobrazována formou tabulky.

5.2.2 Časová náročnost

Jelikože je tato aplikace a její implementace pěvně provázáná s analýzou rozhraní API systému dodavatele, nelze pro ni samostatně časovou náročnost vyjádřit. Ta se však promítla do vyjádření celkové časové náročnosti předchozího zadání (viz podkapitola 5.1).

5.2.3 Analýza a návrh

Podobu aplikace (uživatelské rozhraní a strukturu výkonného kódu) do značné míry určilo samotné API rozhraní dodavatelského systému, jehož analýza tak byla pro implementaci zásadní. Vzhledem k tomu, že aplikace měla sloužit spíše jako interní nástroj, nebyly kladeny požadavky na její robustnost ani kvalitu uživatelského rozhraní, jehož podobu jsem převzal z ostatních aplikací stejného druhu pro jiné dodavatelské systémy.

Pro vytvoření této webové aplikace jsem zvolil platformu ASP.NET Web Forms, která byla použita pro aplikace tohoto druhu i v případě všech předchozích modulů.

5.2.4 Implementace

Prezentační část aplikace se sestává pouze z jedné webové stránky. Výkonný kód psaný v jazyce C# .NET je uložen v samostatném souboru vztahujícím se k této stránce (takzvaný *code-behind*). Důraz na jeho architekturu či rozvrstvení zde není kladen vzhledem k účelu této aplikace.

Pro přenos dat mezi klientskou a serverovou částí aplikace jsem použil komponenty platformy ASP.NET Web Forms. Pro zadání uživatelského vstupu (v tomto případě vstupních parametrů požadavků) jsem použil `TextBox`. Pro zobrazení dat, navrácených z rozhraní API v odpovědích na požadavky, jsem použil `Repeater`, který obecně slouží pro reprezentaci dat v kolekcích, a v něm vnořený `GridView` pro zobrazení dat v tabulce.

Abych zjednodušil implementaci volání požadavků a usnadnil zároveň jejich případnou modifikaci, vytvořil jsem generickou metodu, která v rámci každého volání požadavku obstarává autentizaci a zpracovává navrácený výsledek. Tím se mi zároveň podařilo do značné míry redukovat duplicitní kód. Dílčí metody představující volání jednotlivých požadavků tak pouze specifikují URL, objekt reprezentující očekávaný výstup, typ HTTP požadavku a případná data zasílána v jeho těle. Výpis 1 zachycuje využití této generické metody při implementaci požadavku na vytvoření *nákupního košíku*.

```
protected async Task CreateCart()
{
   var createCartRequest = new CreateCartRequest
   {
      DomainId = DomainTb.Text
   };
   var result = await MakePostApiCall<CartResult>("/carts", createCartRequest);
   //display result
}
```

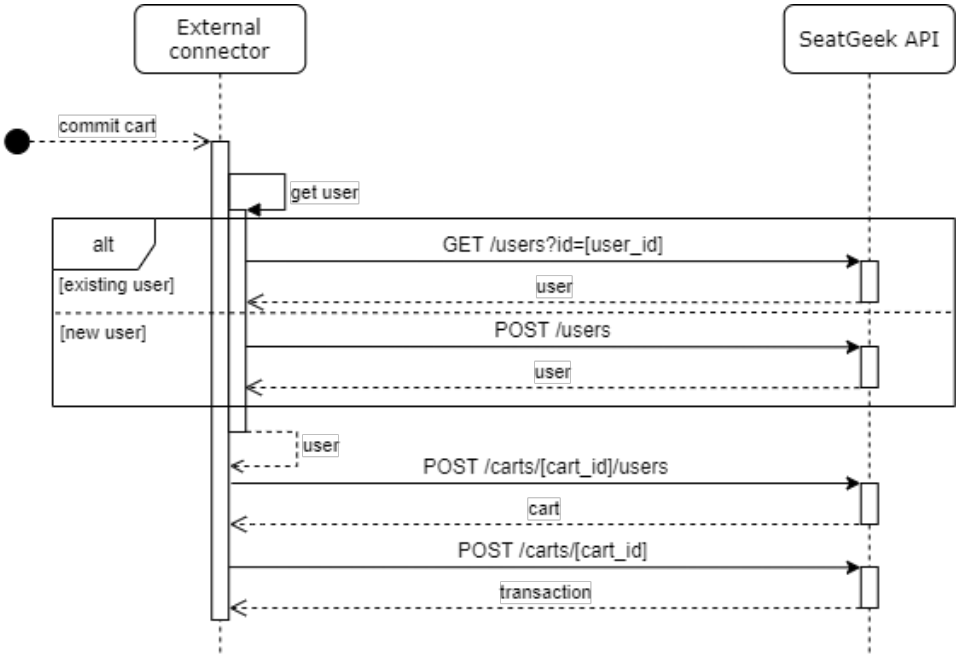
Výpis 1: Využití generické metody

V procesu testování jsem u odpovědí na požadavky týkající se dostupnosti sedadel pro dané představení narazil na problém s vykreslováním příliš objemných dat. Situaci jsem se po konzultaci rozhodl řešit omezením počtu vykreslovaných záznamů naráz. Do uživatelského rozhraní jsem tak navíc přidal textová pole pro zadání počtu záznamů k přeskočení a vypsání.

5.3 External Connector

5.3.1 Specifikace požadavků

Knihovna mající za cíl zprostředkování a usnadnění komunikace s veřejným webovým rozhraním API dodavatele. Knihovna svému konzumentu zprostředkuje zjednodušené rozhraní pro komunikaci se systémem dodavatele skrze volání metod v jazyce C# .NET. Na vstupu bude přijímat pouze nezbytně nutné informace k provedení daného požadavku, a konzumenta tak odstíní od implementačních detailů. Dále bude úkolem knihovny sjednodit a obsluhovat operace společné pro všechna volání požadavků na rozhraní API. Mezi tyto operace patří mimo jiné autentizace, zachytávání výjimek a ošetření chybových stavů či proces serializace a deserializace. Knihovna bude distribuována skrze *NuGet package manager*.



Obrázek 5: Sekvenční UML diagram External Connector

5.3.2 Analýza a návrh

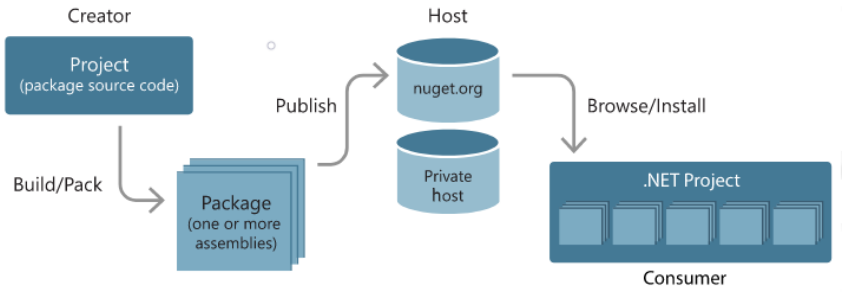
Díky aplikaci *Integration Tester* (viz podkapitola 5.2) jsem v době zpracovávání tohoto zadání již měl otestovanou komunikaci s webovým rozhraním API dodavatele a zároveň jsem měl implementovaný výkonný kód v jazyce C# .NET, který tuto komunikaci zajišťoval. Tato knihovna však kromě samotného zprostředkování komunikace měla za úkol komunikaci co nejvíce zjednodušit, a odstínit tak svého konzumenta od co největšího množství implementačních detailů.

Příkladem je potvrzení objednávky. Zatímco na úrovní aplikace *Integration Tester* je potřeba nejprve nalézt, či vytvořit uživatele, přiřadit jej k objednávce a teprve pak provést její dokončení, konzumentu této knihovny k tomu mělo postačit pouze zavolání jedné metody pro dokončení objednávky, která vnitřně provede veškeré potřebné operace. Tuto metodu zachycuje sekvenční UML diagram na obrázku 5.

Pro vytvoření knihovny jsem zvolil framework .NET Standard 2.0.

Nuget Package Manager

Knihovna měla být distribuována tak, aby ji bylo možné konzumovat v různých, nezávislých částech projektu. *Package manager* je v obecném pojetí nástroj pro sdílení a konzumaci znovupoužitelného kódu. NuGet package manager [16] je jedním z nich. Samotný kód je přenášen ve formě takzvaného *NuGet balíčku*, který lze vnímat jako archív sdružující zkompilovaný kód (soubory .dll) a další data, mezi něž se řadí také informace o jeho verzi. Princip distribuce těchto balíčků zachycuje obrázek 6.



Obrázek 6: Princip distribuce skrze NuGet package manager [16]

V projektu London Theatre Direct dochází k tvorbě a distribuci NuGet balíčků v prostředí Microsoft Azure DevOps. K jejich vytváření a nasazování nových verzí dochází automatizovaně s pomocí CI/CD metodik (viz podkapitola 2.2.1 na straně 15).

OpenAPI specifikace

Aplikace tohoto typu jsou často generovány spíše než vytvářeny manuálně. K tomu je však zapotřebí, aby webové rozhraní API, pro něž je aplikace tohoto druhu vytvářena, poskytovalo OpenAPI specifikaci. Ta zpřístupňuje standardní rozhraní pro práci s danou službou a usnadňuje její pochopení s minimální znalostí implementačních detailů. Podstatné však je, že díky této specifikaci lze generovat dokumentace a také samotný zdrojový kód zajišťující komunikaci se systémem. Takto vygenerovaný kód je sice často dosti objemný a špatně čitelný, zároveň však šetří spoustu nákladů spjatých s manuální implementací. Vzhledem k účelu aplikace, která má především poskytnout rozhraní pro přístup k externímu systému, zde však samotná čitelnost či úhlednost kódu není až tak podstatná. Dodavatelským systémem SeatGeek však tato specifikace poskytována není, tudíž bylo potřeba aplikaci vyvinout bez těchto nástrojů.

5.3.3 Implementace

Z aplikace *Integration Tester* (viz podkapitola 5.2) jsem převzal výkonný kód obsluhující komunikaci s webovým rozhraním API dodavatele. Musel jsem v něm však účinit několik zásadních úprav. Ta první se týkala reprezentace výstupních dat. Zatímco *tester* data po zpracování přímo propisuje do uživatelského rozhraní k zobrazení, tato knihovna měla mít za úkol data pouze předat k dalšímu zpracování konzumentem. Dále bylo potřeba zvýšit míru zapouzdření a samotné rozhraní pro komunikaci s webovým rozhraním API na výstupu z této knihovny zjednodušit (viz podkapitola 5.3.2 věnována analýze této aplikace).

Všechny metody vztahující se k volání jednotlivých požadavků rozhraní API jsem oddělil do samostatné třídy, které jsem nastavil modifikátor přístupu `internal`, aby tak byla přístupná pouze v rámci této knihovny. Veřejné metody pak zpravidla zapouzdřují volání jedné či více těchto metod.

5.4 Performances Mapper

5.4.1 Specifikace požadavků

Webová aplikace sloužící k mapování dat získaných ze systému dodavatele na interní záznamy. Konkrétně se jedná o data vztahující se k představením, jež jsou ukládány v perzistentním úložišti. Po úspěšné autorizaci v dodavatelském systému uživatelské rozhraní uživateli umožní vybrat událost, pro niž bude chtít mapování představení provést. Dále uživatel zvolí záznam z interního úložiště, jež reflektuje danou událost. Bude moci zároveň zadat časový rozsah, a provést tak mapování představení dané události pouze pro omezený časový úsek. Aplikace uživateli zobrazí náhled, aby tak měl možnost podrobně zkontrolovat jednotlivé operace, k nimž dojde.

5.4.2 Analýza a návrh

Pro vytvoření této webové aplikace jsem zvolil platformu ASP.NET Web Forms, která byla použita pro aplikace tohoto druhu také v případě všech předchozích modulů. Pro získání potřebných údajů k představením, a s tím spojenou komunikaci s webovým rozhraním API dodavatele, jsem využil knihovnu *External Connector* (viz podkapitola 5.3).

Uživatelské rozhraní

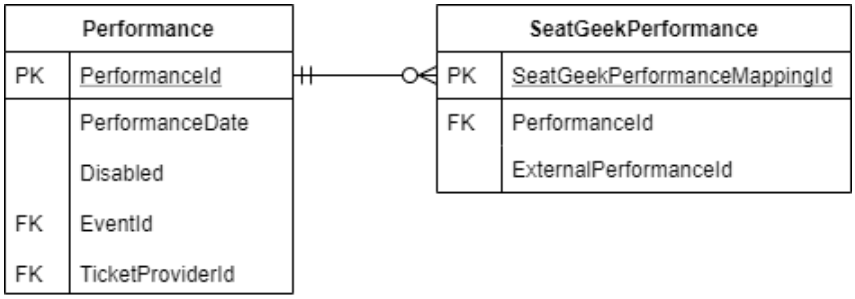
Dříve než jsem začal se samotnou implementací aplikace, jsem si musel rozmyslet podobu uživatelského rozhraní. Bylo potřeba zajistit prostředí dostatečně intuitivní na to, aby uživatel byl schopen provést mapování snadno a přehledně v několika krocích. Skicu s výslednou podobou zachycuje obrázek 12 (strana 46, příloha B).

Perzistentní úložiště

V interním úložišti, jimž je SQL databáze, pro záznamy o představeních existuje samostatná entita. Dále je zde několik entit představujících data o představeních v kontextu daného dodavatelského systému. Mezi těmito entitami existuje vazba. Musel jsem tak nejprve provést databázové úpravy a přidat zmíněnou tabulku představující data o představeních v dodavatelském systému SeatGeek. Situaci znázorňuje relační model na obrázku 7.

S těmito daty měla aplikace provádět následující operace:

- V databázi je evidován pouze záznam reprezentující data z externího systému, nikoliv však záznam pro obecnou entitu představení. Dojde ke vložení tohoto záznamu.
- Záznam pro obecnou entitu představení v databázi existuje, je však provázán se záznamem reprezentujícím data z externího systému pro jiné představení. Dojde k zakázání tohoto záznamu.
- Oba záznamy v databázi existují a jsou validní, chybí však vazba mezi nimi. Dojde k vytvoření této vazby.



Obrázek 7: Relační model Performances Mapper

- Data o představení jsou konzistentní, a není tak potřeba provést žádnou operaci.

5.4.3 Implementace

Dříve než jsem začal se samotnou implementací, musel jsem provést zmíněné databázové úpravy. Přístup k databázi je v projektu zajištěn technikou objektově relačního mapování (viz podkapitola 3.3.4 na straně 23). V odděleném repozitáři zde existuje několik datových modelů knihovny Entity Framework, které jsou distribuovány skrze *Nuget package manager*. Poté co jsem provedl změny v samotné databázi, jsem upravil také tyto datové modely. Distribuce NuGet balíčků probíhá s využitím CI/CD metodik v prostředí Microsoft Azure DevOps (viz podkapitola 2.2.1 na straně 15).

Po schválení změn tak došlo k automatizovanému vydání nové verze, která již zahrnovala mnou provedené změny. Poté již stačilo pouze aktuální verzi knihovny s příslušným datovým modelem referencovat v mé aplikaci a mohl jsem s databází začít pracovat.

Výkonný kód psaný v jazyce C# .NET se nachází opět v takzvaném *code-behind* této stránky stejně jako tomu bylo v případě aplikace *Integration Tester* (viz podkapitola 5.2). Poté co uživatel zvolí vstupní parametry a potvrdí volbu zobrazení náhledu mapování, dojde k načtení dat z perzistentního úložiště skrze získaný datový model. Nad těmito daty je dále prováděna analýza potenciálních operací s nimi. Na základě zmíněných pravidel je pak každému z načtených představení přiřazena operace, která s ním má být vykonána. Tyto informace jsou propsány do uživatelského rozhraní. K tomu jsem využil komponentu platformy ASP.NET Web forms `ListView`, která obecně slouží pro práci s daty v kolekcích.

V případě potvrzení náhledu dojde k znovunačtení vstupní dat a provedení operací s nimi.

5.5 Inventory Synchronizer

5.5.1 Specifikace požadavků

Podstatou této aplikace je zápis a aktualizace dat týkajících se dostupnosti sedadel jednotlivých představení získávaných z webového rozhraní API systému dodavatele. Data budou ukládána v perzistentním úložišti, které bude tvořit formu mezipaměti pro zrychlení a zefektivnění procesu načítání dostupnosti pro jednotlivá představení. Aplikace bude na bázi služby Azure Service Bus přijímat požadavky k synchronizaci. Ty se budou týkat buď konkrétního představení, či celé události.

5.5.2 Analýza a návrh

O této aplikaci se obecněji zmiňuji v podkapitole 3.2.3 na straně 20. Motivací k jejímu vzniku je snížení závislosti na externím systému, který může mnohdy požadavky vyřizovat pomaleji, čímž negativně ovlivňuje celkový uživatelský dojem. Dále se jedná o výkonnostní optimalizace, jelikož z častého kontaktování webového rozhraní API mohou plynout výkonnostní problémy nejen na straně systému dodavatele, ale i v samotné aplikaci pracující s ním.

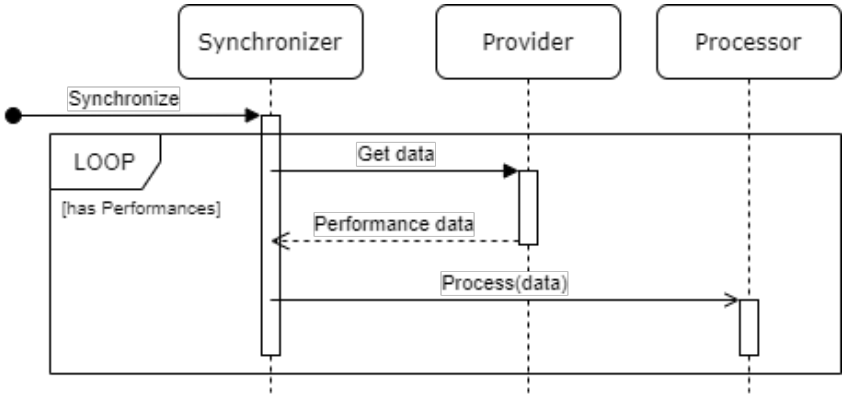
Aplikace měla požadavky k synchronizaci přijímat na bázi fronty, k čemuž jsem použil službu Azure Service Bus (viz podkapitola 3.3.5 na straně 23). Nasazení této služby mělo přinést především možnou škálovatelnost. Již při samotném vývoji aplikace jsem tak věděl, že dojde k jejímu nasazení v několika instancích, z nichž každá z nich bude zpracovávat požadavky z téže fronty. Aby aplikace mohla naslouchat příchozím požadavkům, bylo potřeba zajistit, aby byla trvale spuštěna. K tomu se využívá spuštění aplikace v režimu Windows služby. V aplikacích tohoto druhu se v případě předchozích modulů používá externí knihovna Topshelf [17]. Rozhodl jsem se ji použít i zde. Konzolová aplikace s touto knihovnou kromě běžného režimu spuštění v systémové konzoli může být spuštěná právě jako Windows služba.

Pro potřeby interakce se službou Azure Service Bus, v případě této aplikace přijímání požadavků z fronty, byla v projektu London Theatre Direct zřízená knihovna, která zapouzdřuje nutné operace a svému konzumentu zpřístupňuje zjednodušené rozhraní.

Způsob zpracování požadavků

Musel jsem zvážit, jaký typ požadavků bude aplikace nejčastěji přijímat a jak bude docházet ke zpracování a aktualizaci dat k jednotlivým představením. Zpočátku jsem chtěl provádět toto zpracování až po načtení veškerých potřebných dat k představením. Zde jsem si však uvědomil, že je tento způsob dosti neefektivní. Zároveň v případě selhání aplikace při načítání vstupních dat by došlo k zahození všech výsledků bez provedení jakékoliv aktualizace.

Rozhodl jsem se proto svůj přístup změnit a každé zpracovávané představení si představit jako oddělenou iteraci běhu této aplikace, v niž může dojít k načtení dat a zároveň i k jejich následnému zpracování. Podstatné bylo si uvědomit, že k samotnému zpracovávání musí do-

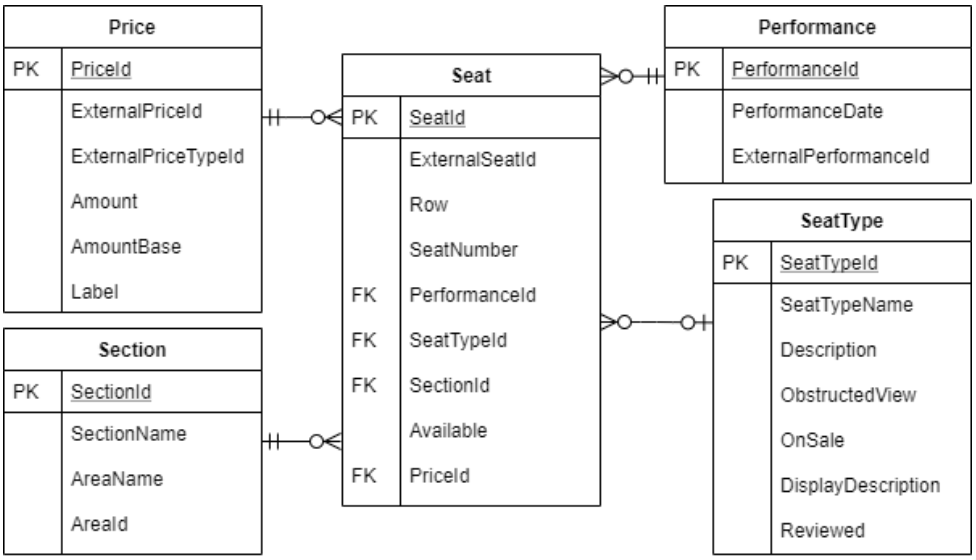


Obrázek 8: Sekvenční UML diagram Inventory Synchronizer

cházet asynchronně, aby tak nedocházelo k blokování vnější metody pro synchronizaci všech představení, která tak může spustit další iteraci, aniž by musela čekat na dokončení té původní. Navržený proces synchronizace zachycuje sekvenční UML diagram na obrázku 8.

Zpracovávaná data

Jak již zmiňuje specifikace požadavků pro tuto aplikaci (viz podkapitola 5.5.1), data o dostupnosti sedadel měla být z API rozhraní ukládána do perzistentního úložiště. Tím je v projektu London Theatre Direct primárně SQL databáze. Musel jsem proto prvně navrhnout strukturu těchto dat. Do této analýzy jsem zahrnul již dříve implementované moduly a strukturu potřebnou pro tento systém jsem navrhnul v korelaci s nimi. Po vyčlenění jednotlivých entit jsem navrhl vztahy mezi nimi. Výsledný stav zachycuje relační model na obrázku 9.



Obrázek 9: Relační model Inventory Synchronizer

Seat – představuje konkrétní sedadlo v hledišti. `Row` a `SeatNumber` slouží k jednoznačné identifikaci jeho polohy vůči ostatním sedadlům v dané sekci hlediště (viz *Section*). `Available` informuje o dostupnosti a `ExternalSeatId` reprezentuje identifikátor daného sedadla v systému dodavatele.

Performance – představuje dané představení, kterého se data o dostupnosti sedadel týkají. `PerformanceDate` informuje o datu konání a `ExternalPerformanceId` představuje identifikátor daného představení v externím systému.

SeatType – realizuje typ sedadla. Reprezentuje skupinu sedadel mající společné vlastnosti, které však daná sedadla spíše odlišují od jiných. Z toho plyne, že ne každé sedadlo musí být součástí této skupiny[1]. Těmito typy pak zpravidla jsou prémiové sedadla či naopak ta s omezeným výhledem. `Description` popisuje zmíněné vlastnosti. `ObstructedView` informuje o případném zhoršeném výhledu. `OnSale` značí, zda-li je tento typ sedadel dostupný k prodeji a `Reviewed` slouží pro interní účely, jelikož každý z typů sedadel, jež jsou z externího systému nabízeny, musí být navíc zkontrolován a schválen členy týmu společnosti London Theatre Direct.

Price – reprezentuje cenovou hladinu. Téměř pro každé představení jsou sedadla nabízená v různých cenových hladinách. Na rozdíl od typu proto platí, že každému sedadlu musí být přiřazena cena. Cenové hladiny jsou seskupovány podle typu (`ExternalPriceTypeId`). `ExternalPriceId` představuje indentifikátor cenové hladiny v rámci externího systému dodavatele. `Amount` popisuje cenu včetně poplatků. `AmountBase` pak představuje nominální hodnotu.

Section – reprezentuje sekci, do niž jsou sedadla v hledišti členěna. Opět zde platí, že každé sedadlo náleží právě jedné sekci. V projektu následně interně dochází k mapování na jednotnou entitu představující tuto sekci napříč všemi dodavatelskými systémy. `SectionName` informuje o názvu sekce v rámci dodavatelského systému a `AreaName` spolu s `AreaId` již představují údaje vztahující se k společné entitě v rámci projektu, na niž jsou sekce jednotlivých dodavatelských systémů mapovány.

[1] Nemá přiřazený typ.

5.5.3 Implementace

Zadání jsem zpracoval jako konzolovou aplikaci. Knihovna Topshelf, jež je referencována skrze *NuGet package manager*, umožňuje, aby tato aplikace byla spouštěná jak v režimu Windows služby, tak i skrze systémovou konzoli.

Přijetí požadavku

Knihovna Topshelf definuje třídu `HostFactory` a její metodu `Run`, která je volaná metodou `Main` konzolové aplikace. V rámci volání metody `Run` jsou předávání delegáti na funkce pro obsluhu událostí spuštění a zastavení běhu aplikace. Samotná implementace těchto funkcí se nachází v samostatné třídě. Dále jsou zde definovány informace vztahující se k instalaci Windows služby. Mezi ty patří především název, popis a pravidla spuštění služby.

Funkci volané při spuštění jsou na vstupu předány kromě instance třídy `HostControl`, která umožňuje řídit běh aplikace v režimu konzole, také přijaté parametry. Na základě těchto parametrů funkce rozhodne, zda aplikace běží v režimu konzole či Windows služby. V prvním případě je požadavek přijat z parametrů předaných skrze konzoli a po jeho zpracování je aplikace ukončena[2]. V opačném případě je aplikace nastavena tak, aby přijímala příchozí požadavky skrze frontu Azure Service Bus.

Pro interakci s touto službou využívám interní knihovnu, jež je touto aplikací referencována opět skrze *NuGet package manager*. Tato knihovna definuje třídu `ServiceBusReceiver`, které je při inicializaci předáno spojení ke konkrétní instanci služby a další nastavení, mezi které patří maximální počet souběžně běžících vláken, v nichž jsou požadavky zpracovávány, či limit nastavující dobu, po které je požadavek uvolněn zpět ke zpracování jiným příjemcem, pokud toho aktuální příjemce není schopen. Nad takto vytvořenou instancí třídy je volaná metoda `BeginReceiveFromQueue`, které již pouze stačí na vstupu předat název fronty, z které aplikace požadavky přijímá, a delegáty na funkce pro zachycení událostí přijetí zprávy a detekci vzniku případné výjimky.

Zpracování požadavku

Přijatý požadavek v podobě deserializovaných parametrů ze systémové konzole či zprávy zaslané skrze službu Azure Service Bus je dále zpracováván. Pokud se jedná o požadavek přijatý skrze konzoli, dochází k jeho zpracování synchronně. Skrze frontu jsou však takto zpracovávány pouze požadavky týkající se konkrétního představení. Na výsledek zpracování požadavku k synchronizaci celé události však aplikace *nečeká* vzhledem k jeho časové náročnosti. V informatice pro toto chování existuje ustálený pojem *fire and forget*.

V jádru se zpracování požadavku k synchronizaci dat týkajících se jednoho představení neliší od zpracovávání dat pro všechna představení celé události. V obou případech je volána metoda `SynchronizePerformances`, které jsou na vstupu předány identifikátory představení, pro něž má

[2]Dojde k zavolání metody Stop třídy HostControl, čímž se vyvolá druhá ze zmíněných událostí.

k synchronizaci dojít. Pokud jsou zpracovávána data pro jedno představení, je pouze ověřována správnost identifikátoru přijatého v požadavku. V opačném případě jsou prvně dle získaného identifikátoru události načtena odpovídající představení, jejichž identifikátory jsou dále předány této metodě. Na základě předaných identifikátorů metoda `SynchronizePerformances` načte potřebná data o představeních z databáze (viz relační model na obrázku 7 na straně 33).

Data načtená z databáze jsou dále zpracovávána. V tomto místě vstupuje do hry problematika paralelního načítání a zpracovávání dat z dodavatelského systému (viz podkapitola 5.5.2). K dosažení této funkcionality jsem se rozhodl využít techniky zvané *lazy load*. V jazyce C# .NET k tomu slouží konstrukce `yield return` nad rozhraním `IEnumerable`. Ta na pozadí vytváří stavový automat, který si pamatuje aktuální pozici v právě procházené kolekci, a umožňuje tak při procházení kolekce skrze enumerátor postupné uvolňování zpracovávaných prvků.

Výpis 2 zachycuje příklad použití. S pomocí příkazu `foreach` metoda `ProcessData` přistupuje skrze volání metody `GetEnumerator` postupně k prvkům kolekce navrácené z metody `GetData`. První hodnota je tak zpracována dříve, než dojde k načtení druhé hodnoty.

```
IEnumerable<object> GetData()
{
   yield return ...;
   yield return ...;
}
void ProcessData()
{
   foreach(int number in GetData())
   {
      //process
   }
}
```

Výpis 2: Příklad užití konstrukce yield return

Metodu `GetPerformancesData` (viz výpis 3) lze považovat za ekvivalent k metodě `GetData` z předchozího příkladu. Jelikož však v metodě dochází k načítání dat z webového rozhraní API systému dodavatele, bylo vhodné zajistit, aby metoda byla navíc asynchronní. Důvodem je zabránění blokování vlákna zajišťujícího zasílání HTTP požadavku, které tak může být využito pro jiné operace. Pokud by totiž metoda byla synchronní, vlákno by touto vstupně-výstupní operací bylo blokováno, dokud by nedošlo k jejímu zpracování.

V jazyce C# 7.3 však neexistuje nativní podpora pro použití konstrukce `yield return` v asynchronních metodách. Tuto funkci zvanou *async streams* přináší teprve C# 8.0, který jsem však v daných podmínkách použít nemohl.

Situaci se mi podařilo vyřešit s pomocí knihovny AsyncEnumerator [18]. Knihovna přináší rozhraní `IAsyncEnumerable` a pomocnou třídu `AsyncEnumerable`. Té lze v konstruktoru defi-

novat asynchronní lambda výraz, který reprezentuje tělo enumerátoru. Metoda `ReturnAsync` nahrazuje volání `yield return`.

```
public IAsyncEnumerable<SynchronizationData> GetPerformancesData(IEnumerable<
    PerformanceToSynchronize> performances)
{
   return new AsyncEnumerable<SynchronizationData>(async yield =>
   {
      foreach (var performance in performances)
      {
         var data = await GetPerformanceData(performance);
         await yield.ReturnAsync(data);
      }
   });
}
```

Výpis 3: Načtení dat o dostupnosti z dodavatelského systému

Data načtená ze systému dodavatele jsou postupně zpracovávána metodou `ForEachAsync`, která je další z těch, které tato knihovna nabízí. Použití metody `ForEachAsync` zachycuje výpis 4. Jedná se o část těla metody `SynchronizePerformances`. Jednotlivé požadavky ke zpracování jsou po načtení potřebných dat prováděny asynchronně, čímž může v cyklu, v němž jsou data načítána, dojit k načtení dat pro další představení dříve, než je zpracován původní požadavek (viz podkapitola 5.5.2 věnující se analýze této problematiky). Volání metody `Task.WhenAll` po načtení všech potřebných dat pak zajistí, že aplikace nebude pokračovat ve vykonávání zbytku metody `SynchronizePerformances` dříve, než dojde ke zpracování všech požadavků.

```
var tasks = new List<Task<DataSynchronizationResult>>();

await synchronizationDataProvider.GetPerformancesData(performances)
.ForEachAsync(data =>
{
   tasks.Add(synchronizationProcessor.ProcessData(data).ContinueWith(result =>
   {
      return result.Result;
   }));
})

var results = await Task.WhenAll(tasks).ConfigureAwait(false);
```

Výpis 4: Synchronizace načtených dat o dostupnosti

5.6 Ostatní

Zadání úkolů, jež popisuji výše, tvořila většinu náplně mé práce při absolvování odborné praxe ve společnosti ICT Capital. Vzhledem k objemu práce, kterou implementace modulu pro připojení dodavatelského systému představuje, byla práce rozdělena v rámci několikačlenného týmu, s kterým jsem úzce spolupracoval. Neměl jsem tak možnost podílet se přímo na implementaci všech částí tohoto modulu, s řadou z nich jsem se však setkal v procesu testování, do kterého jsem byl zahrnut. Podílel jsem se také na zákaznické podpoře, k niž se zároveň vztahuje celá řada drobnějších úkolů, které mi byly zadány, avšak neměly přímou spojitost s moduly pro připojení dodavatelského systému. Nezanedbatelnou část svého působení ve společnosti jsem strávil také studiem projektu London Theatre Direct a všech jeho náležitostí.

6 Zhodnocení

Čas strávený ve společnosti ICT Capital a především účast na projektu London Theatre Direct mi přinesla nespočet cenných zkušeností a znalostí. Uvědomil jsem si, že tak jako nejspíše i v jiných odvětvích, ani v případě vývoje software nelze vždy ideálně následovat teoretické modely, jenž často nepředpokládají vliv mnoha vnějších aspektů, které mohou vést k jejich narušení. Patří mezi ně nejen finance, ale i požadavky zákazníka. Právě zakázkový vývoj vyžaduje obzvlášť velkou dávku flexibility ze strany společnosti zaštiťující technické řešení daného projektu.

Jak již bylo zmíněno v úvodu, společnost na všech svých aktivních projektech využívá metod agilního řízení projektů. Konkrétně se jedná o Scrum a Kanban. Nelze však říci, že by tyto metody byly aplikovány dogmaticky. Společnost využívá a kombinuje pouze vybrané techniky, které považuje za prospěšné. Takto nastavený proces vývoje (viz podkapitola 2.2 na straně 15) na mě působí přirozeně a po začlenění do týmu jsem si na něj velmi rychle zvykl.

Největší přínos absolvování této praxe však vnímám v nabytých technických znalostech. Významně jsem si rozšířil znalosti týkající se programovacího jazyka C# .NET a celé platformy .NET. Pracoval jsem na aplikacích zpracovávajících velký objem dat, u niž byly často zároveň kladeny vysoké výkonnostní nároky, což mě mnohdy vedlo k opakovanému přepisu zdrojového kódu a uvědomení si častých chyb, které jsem do té doby dělal, avšak jejich důsledky nebyly až tak znatelné.

6.1 Uplatněné znalosti a dovednosti

Vzhledem k podobnosti mé náplně práce s oborem, jež studuji, jsem měl možnost uplatnit mnoho nabytých teoretických a praktických znalostí a dovedností získaných za dobu mého dosavadního studia.

Jelikož téměř všechny aplikace v rámci projektu London Theatre Direct pracují s daty uloženými v SQL databázi, ocenil jsem základy získané absolvováním předmětů *Úvod do databázových systémů* a *Databázové a informační systémy*.

Pro pochopení principů vývoje software a návrh možného řešení jsem zužitkoval znalosti týkající se obecných základů algoritmizace a návrhu architektury software, jež mi byly předány v předmětech *Algoritmy I* a *II* a *Vývoj informačních systémů*.

Zasádní pro mě zcela jistě byly znalosti nabyté absolvováním všech předmětů týkajících se programování, které mi mimo jiné přinesly obecný přehled a shopnost programátorsky uvažovat. Jedná se o předměty *Programování I* a *II* a *Programovací jazyky I*. Schopnosti a dovednosti získané absolvováním předmětů *Programovací jazyky II* či *Architektura technologie .NET* pro mě byly obzvlášť přínosné vzhledem k vazbě na platformu .NET, s niž jsem se dennodenně ve svých zadáních potýkal.

6.2 Scházející znalosti a dovednosti

Při vypracovávání úloh jsem velice brzy narazil na omezené znalosti a dovednosti týkající se platforem pro vývoj webových aplikací. O platformě ASP.NET Web Forms jsem dosud měl pouze velmi omezené teoretické znalosti, které však nebyly podložené praktickou zkušeností. Tento nedostatek se jistě projevil i na časové dotaci, s kterou jsem vybraná zadání řešil.

Brzy jsem také musel dohnat značné rezervy ve znalostech týkajících se verzovacího systému Git a pokročilejších funkcí vývojového prostředí Microsoft Visual Studio.

V neposlední řadě jsem musel zapracovat na prohloubení znalosti jazyka C# .NET a dovednostech práce s některými z externích knihoven, mezi něž se řadí především Entity Framework.

7 Závěr

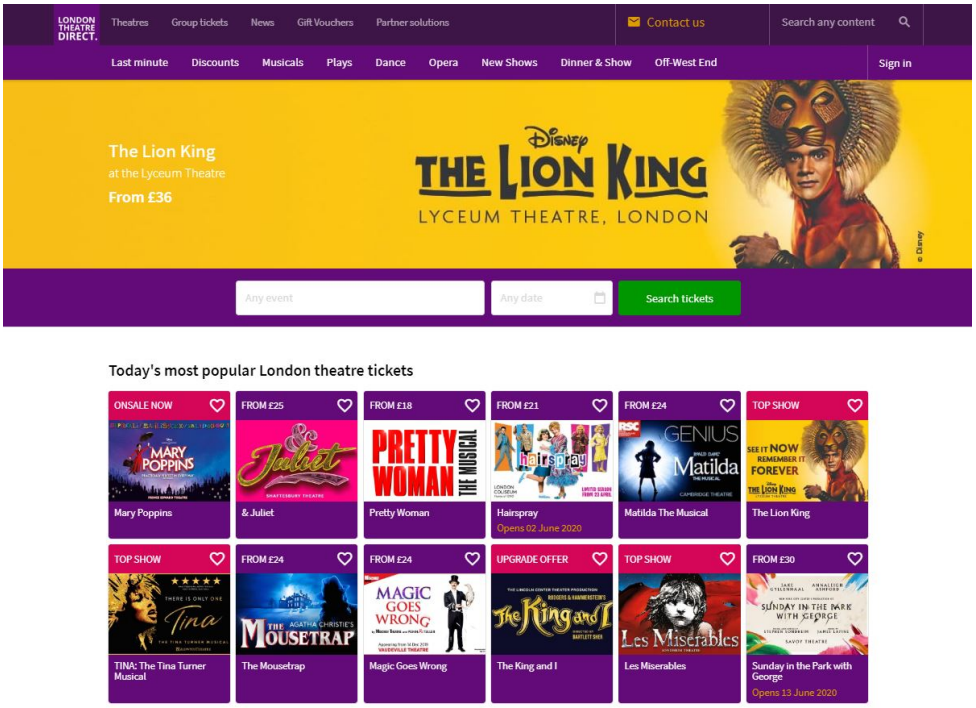
Výsledkem absolvování mé odborné praxe ve společnosti ICT Capital, v niž jsem pracoval na projektu London Theatre Direct, je částečně implementovaný modul pro připojení dodavatelského systému SeatGeek. Práce v několikačlenném týmu zkušených programátorů pracujících na tak rozsáhlém projektu mi přinesla cenné zkušenosti a poznatky, z kterých zcela určitě budu v budoucnu těžit. Uvědomil jsem si, jak zásadní dovedností vývojáře je flexibilita. Do vývoje totiž často vstupují okolnosti, jimž se musí vývojář umět přizpůsobit. Ty si mnohdy žádají provizorní řešení a učí ho přistupovat k situacím individuálně dle aktuální potřeby. Pochopil jsem taky, jak podstatná je správná komunikace mezi členy týmu a pocítil její vliv na kvalitu řízení projektu.

V neposlední řadě mě zkušenost s touto společností utvrdila ve správnosti mého rozhodnutí věnovat se tomuto oboru činnosti i nadále ve své profesi a dodala mi potřebnou motivaci k dalšímu studiu.

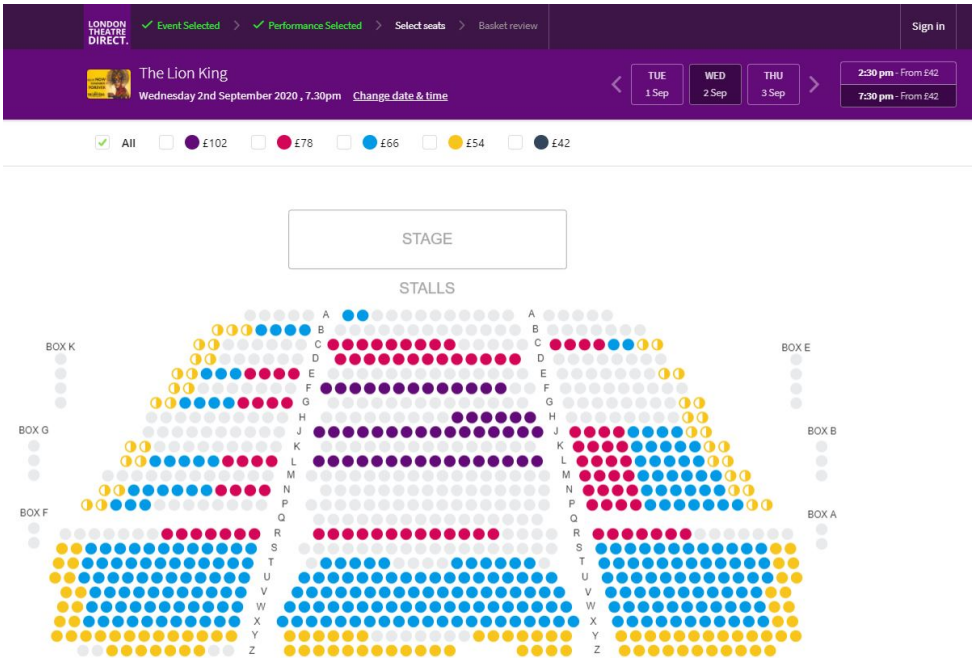
Reference

1. *ITIXO* [online]. ITIXO [cit. 2020-04-30]. Dostupné z: https://itixo.com/.
2. *Azure DevOps* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://azure.microsoft.com/cs-cz/services/devops/.
3. *Git* [online]. Scott Chacon [cit. 2020-04-30]. Dostupné z: https://git-scm.com/.
4. *CI/CD (Continuous Integration/Continuous Deployment)* [online]. SupportPRO [cit. 2020-04-30]. Dostupné z: https://www.supportpro.com/blog/ci-cd/.
5. *RIGANTI* [online]. RIGANTI [cit. 2020-04-30]. Dostupné z: https://www.riganti.cz/.
6. *DotVVM* [online]. RIGANTI [cit. 2020-04-30]. Dostupné z: https://www.dotvvm.com/.
7. *Update Conference Prague* [online]. Update Conference [cit. 2020-04-30]. Dostupné z: https://www.updateconference.net/.
8. *London Theatre Direct* [online]. London Theatre Direct [cit. 2020-04-30]. Dostupné z: https://partners.londontheatredirect.com/.
9. *Documentation* [online]. London Theatre Direct [cit. 2020-04-30]. Dostupné z: https://developer.londontheatredirect.com/documentation.
10. *Microsoft .NET* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://dotnet.microsoft.com/.
11. *ASP.NET Web Forms* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://dotnet.microsoft.com/apps/aspnet/web-forms.
12. *Entity Framework Documentation* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://docs.microsoft.com/en-gb/ef/.
13. *Language Integrated Query (LINQ)* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/programming-guide/concepts/linq/.
14. *Service Bus: Messaging as a service on Azure* [online]. Cloud Academy [cit. 2020-04-30]. Dostupné z: https://cloudacademy.com/blog/service-bus-messaging-as-a-service-on-azure/.
15. *SeatGeek* [online]. SeatGeek [cit. 2020-04-30]. Dostupné z: https://seatgeek.com/.
16. *An introduction to NuGet* [online]. Microsoft [cit. 2020-04-30]. Dostupné z: https://docs.microsoft.com/cs-cz/nuget/what-is-nuget.
17. *Topshelf* [online] [cit. 2020-04-30]. Dostupné z: http://topshelf-project.com/.
18. *Dasync/AsyncEnumerable* [online]. GitHub [cit. 2020-04-30]. Dostupné z: https://github.com/Dasync/AsyncEnumerable.

A Webová stránka London Theatre Direct

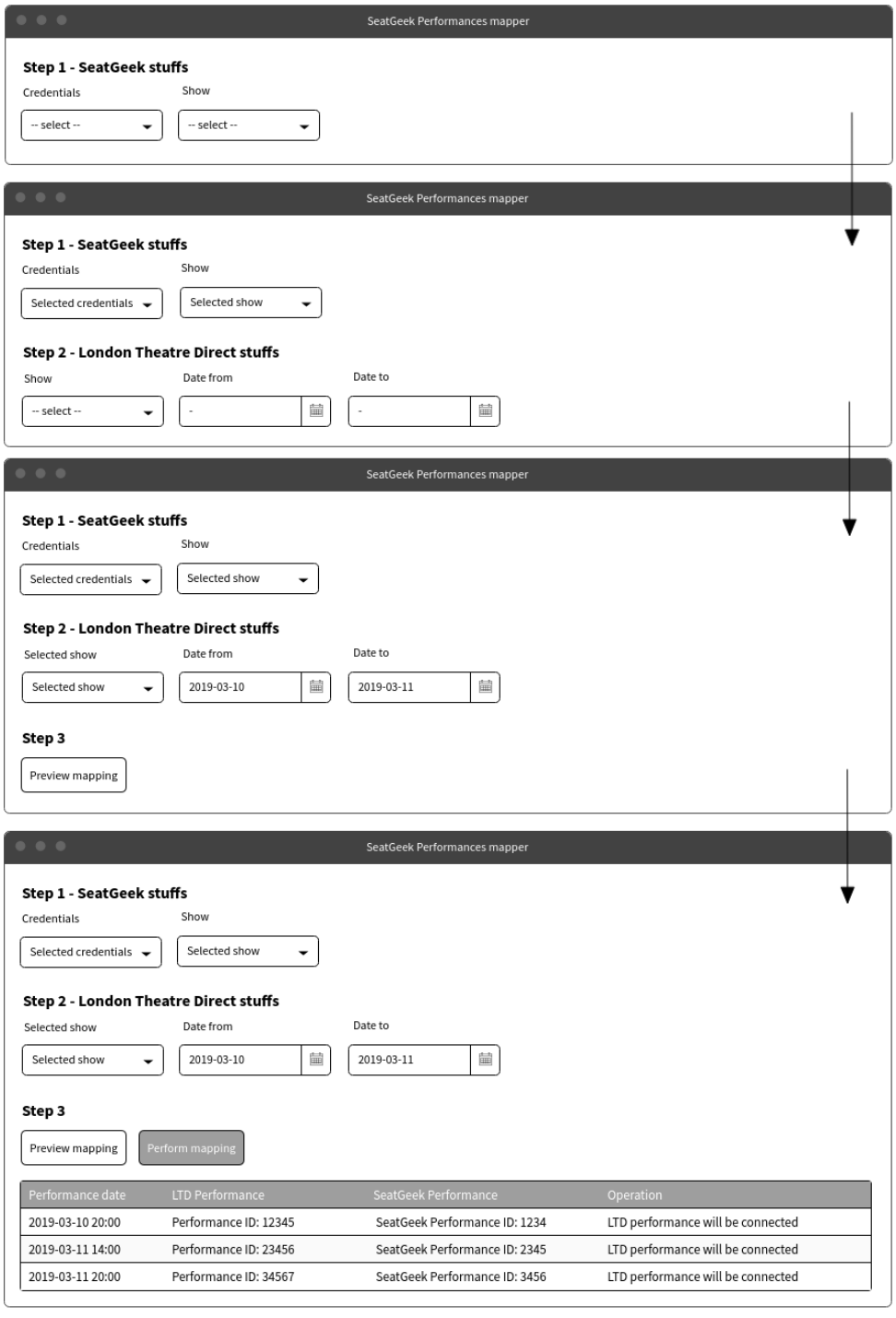


Obrázek 10: Domovská stránka London Theatre Direct



Obrázek 11: Interaktivní plán hlediště

B Uživatelské rozhraní aplikace Performances Mapper



Obrázek 12: Skica uživatelského rozhraní Performances Mapper